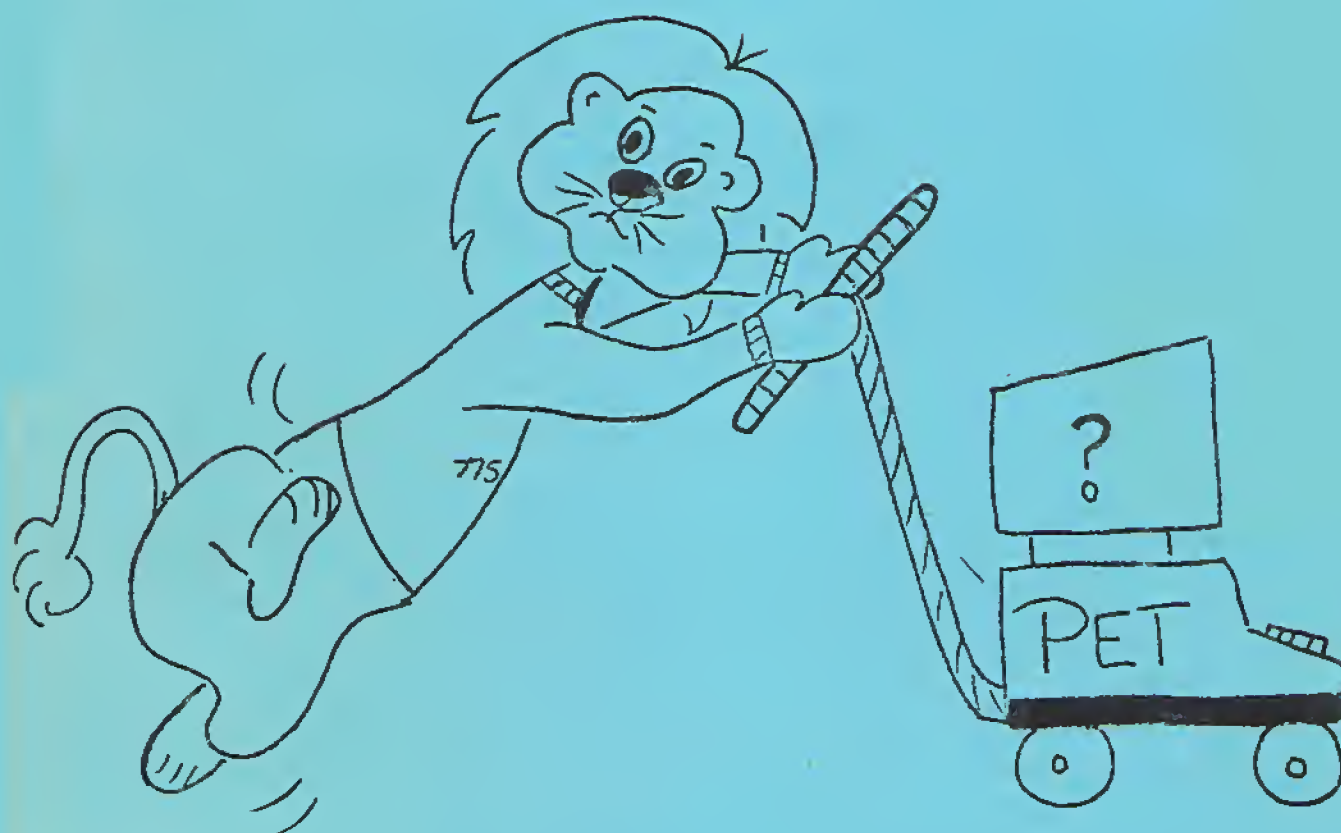


# TIS

TOTAL INFORMATION SERVICES

# Workbook 1



Getting Started with Your PET

## GETTING STARTED WITH YOUR PET

### CONTENTS

I.	INTRODUCTION. . . . .	1-1
A.	Assumptions Made About the User . . . . .	1-1
B.	Exercises . . . . .	1-1
C.	Programming . . . . .	1-2
D.	PET Keyboard and TV Display . . . . .	1-4
E.	Notation Used in This Workbook. . . . .	1-4
F.	BASIC Overview. . . . .	1-6
II.	PET BASIC CALCULATOR MODE . . . . .	2-1
A.	Using Strings in PET BASIC. . . . .	2-2
B.	Numeric and Fractional Values . . . . .	2-2
C.	Conversion of Data. . . . .	2-3
D.	Balance Your Checkbook in Calculator Mode . . . . .	2-4
E.	Reserved Words. . . . .	2-5
F.	Modes of Variables and Constants. . . . .	2-5
III.	INPUTTING A PROGRAM . . . . .	3-1
A.	Blanks. . . . .	3-1
B.	Multiple Program Statements . . . . .	3-1
C.	Typing Mistakes . . . . .	3-2
D.	PET BASIC Commands. . . . .	3-3
E.	The 'STOP' Key and CONT . . . . .	3-7
IV.	GETTING INFORMATION OUT OF YOUR PROGRAM . . . . .	4-1
A.	Output Formats - Numeric Data . . . . .	4-1
B.	Output Formats - Character Strings. . . . .	4-3
C.	Spacing . . . . .	4-3
V.	GETTING INFORMATION INTO YOUR PROGRAM . . . . .	5-1
A.	Design Goals. . . . .	5-1
B.	READ and DATA . . . . .	5-1
C.	INPUT . . . . .	5-2
1.	Floating Point Input. . . . .	5-2
2.	String Input. . . . .	5-6
3.	Integer Input . . . . .	5-11
D.	GET . . . . .	5-12
1.	Numeric GET . . . . .	5-12
2.	String GET. . . . .	5-13

VI.	DATA REPRESENTATION. . . . .	6-1
A.	Largest Numeric Value. . . . .	6-1
1.	Floating Point . . . . .	6-1
2.	Integer. . . . .	6-2
B.	Smallest Numeric Value . . . . .	6-3
1.	Floating Point . . . . .	6-3
C.	Memory Space Used. . . . .	6-4
D.	Number of Significant Digits . . . . .	6-5
E.	Rounding . . . . .	6-6
VII.	USING THE CASSETTE FOR PROGRAM STORAGE . . . . .	7-1
A.	SAVE a Program . . . . .	7-1
B.	VERIFY a Program . . . . .	7-1
C.	LOAD a Program . . . . .	7-3

## I. INTRODUCTION

This workbook gives a series of exercises that show the new user how to use the Commodore PET 2000 computer. The most effective way to use the workbook is to sit down with a PET and do the exercises as they are presented. Enough space has been provided in the workbook for you to add your own examples as you develop them. Later, when you need to refresh your memory on a particular topic, these examples will supply pertinent, meaningful information.

### A. Assumptions Made About the User

Some PET users are comfortable with mathematics. However, this workbook assumes that the majority are not. For that reason, most exercises will use nothing more than high school arithmetic.

This workbook also assumes you know something about the syntax and semantics of the BASIC programming language. If you do not, you should obtain one of the following books:

BASIC Programming, J. Kemeny and T. Kurtz

BASIC by Albrecht, Finke, Brown

What Do You Do After You Hit Return? by Peoples Computer Co.

Basic BASIC by James Coan

Advanced BASIC by James Coan

Read about BASIC syntax; then alternate between this workbook and your text on BASIC. You will learn both BASIC and how to use the PET.

### B. Exercises

Some of the exercises are designed to be done by rote; that is, if you type this, then you should get this response. As you learn more about a feature, the exercises take the form "What would you get if you typed this?" The next level suggests that you try various type-ins.

If you understand the topic, you should be able to anticipate most of the responses. In cases where there are idiosyncrasies, exceptions, and special problems, they are shown as "try-these" exercises.

The last level of exercises is "do-it-yourself," in which you are given guidelines to create a personal example. If you find you cannot construct a personal example, review the previous material and try again.

You will find that you will learn more if you take the time to create or try new things. Be sure to enter the results of both successful and unsuccessful attempts in your workbook so you can have a

convenient reference about what you have done.

Do all the exercises in the workbook. DON'T skip any of them.

C. Programming

Contrary to what some people say, the best way to learn how to program is to program. You can pick up the mechanics of programming by reading books on programming, but that is like reading about moves in chess. Knowing the mechanics of chess does not make you a good chess player. Knowing the syntax of BASIC does not make you a good BASIC programmer.

Just as you can learn chess strategy by reviewing moves made by good chess players, you can learn how to program by reading other people's programs. But, be careful--reading bad code teaches bad habits just as reading good code teaches good habits.

There are several useful tips all programmers should keep in mind. First, if you wish to succeed at programming, you need to develop a consistent style. To succeed at programming means to be able to write reasonable-size codes, quickly and accurately, that are easy to debug, easy to modify, and easy to use. Why is consistent style important? If you use the same approach to different problems, you can:

1. Use pieces or programs elsewhere. This saves time since you don't have to start again from scratch.
2. Understand what you have written some time ago. Consistent style minimizes questions like "What was I thinking about when I tackled this problem?"
3. Reuse documentation from previous programs.
4. Be consistent in error or warning messages.

Second, plan before you code. This is really part of your style or method of solving a problem. One temptation that many programmers succumb to is to "wing it." That means to just sit down at the keyboard and start typing in a program without forethought, planning, or design. DON'T DO IT--not even when you know the problem is trivial. The danger in winging it on a trivial problem is that the program may work and it may work right. But, at some point you will tackle a problem thought to be trivial when it isn't. All successes at winging it will make it that much harder to admit that a particular problem is not trivial. In addition, all the winging-it successes will develop no skills at systematically designing a program and will also tempt you to slip back into that mode whenever you think you have another "trivial" problem to solve. Fig. 1 shows a good approach to writing programs.

## INTRODUCTION

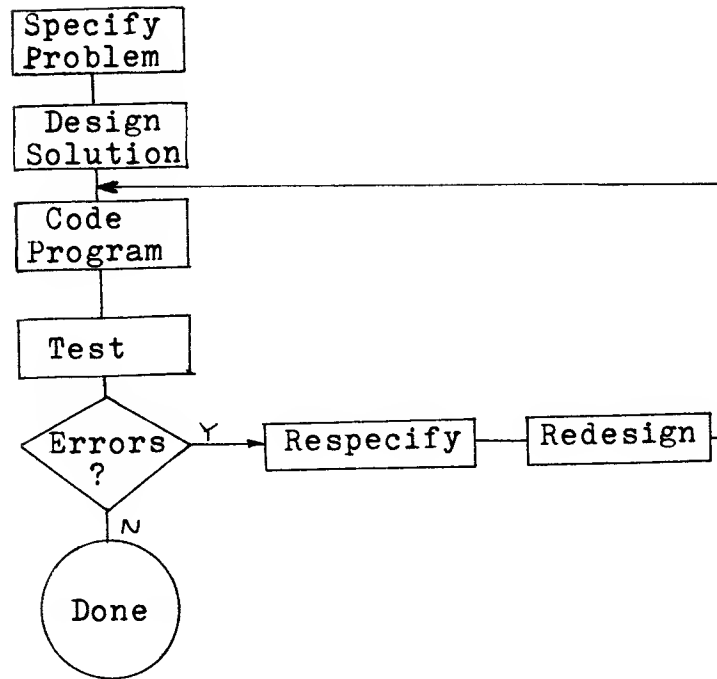


Fig. 1. Program development flowchart.

What is a good way to improve a program? When you have finished a program (that is, you have it working as desired), discard it and start over with the design phase. Why start there? Because now that you have the program working, you know more about the problem, and you can now do a better, more concise, and more effective design. Trying to improve a program by patching up a faulty design might gain you a little, but it may introduce more problems than it removes. Remember, it is never too late to discard a program and start over. Look at your program more like a rough draft--something to learn from. Even though it has a short life, it has served its purpose if you have learned how to do a better job from it.

Many publications have stressed the idea that programming is easy and that anyone can learn to program. This is somewhat misleading. During the last 15 years, we have taught programming in different languages on different machines to people with a variety of backgrounds. Some became programmers; many did not. Our experience is not unique. Just as some people can become good bridge players, piano players, or tennis players, others do not or cannot.

However, you do not have to become an accomplished programmer to enjoy your PET. Preprogrammed cassettes are available from several sources. If you understand some of the BASIC language, you can tailor these programs and make them more personal for your application.

## GETTING STARTED WITH YOUR PET

### D. PET Keyboard and TV Display

The PET keyboard (Fig. 2) and display are the primary input and output devices. To successfully communicate with your PET, you must understand the features of the keyboard and how to use the video display effectively. You can learn this by doing exercises in the manual that Commodore supplied with your PET.

Some of the more important characteristics of the PET keyboard input are:

1. For all nongraphics in BASIC, lower case (unshifted characters) is all that is needed.
2. All lines end with 'RETURN'.
3. The 'DEL' key will delete characters.
4. If there is no keyboard response to the 'STOP' or 'RETURN' keys, power off/on reloads the system and gives a fresh start. Any program entered is lost.
5. To delete the line you are typing, press the 'CLR' key. That deletes the entire line typed so far, and it also erases the entire screen. All previous lines are still in memory.

To use your PET, plug it into a 110 volt AC circuit and turn the power switch on the back of the unit to ON. The first response that you should see on the TV display is

```
R: ***COMMODORE BASIC***  
R: 7167 BYTES FREE  
R: READY
```

All numbers listed for free space reflect an 8K RAM system. If you have a different size memory, the number of free bytes will be different.

### E. Notation Used in This Workbook

We use a consistent notation in this workbook to indicate what is to be typed on the keyboard (T:), what appears on the TV display (R:), and what indicates blanks are to be typed (b). For example:

```
T: info ('RETURN' key)
```

means to type the characters contained on the line after the colon (:) followed by a 'RETURN'.

## INTRODUCTION

R: response

means that the system response to the previous T: line should be a line on the TV.

Blanks are important. They are specified by b. For example:

T: ?"ABbC" ('RETURN' key)

means to type ?, then ", then the letter A, then the letter B, then a space, then the letter C, then " followed by a 'RETURN'.

Now let's run that example all together:

T: ?"ABbC" ('RETURN' key)

R: AB C

The 'RETURN' key must be pressed at the end of each line. We will assume you know that and will not use ('RETURN' key) in any more examples.

The special keys on the PET keyboard can cause some confusion. This workbook identifies the special keys with the notation 'KEYNAME'. 'KEYNAME' means press the named key. The unquoted sequence of characters OTHER means press the five keys O T H E R in succession.

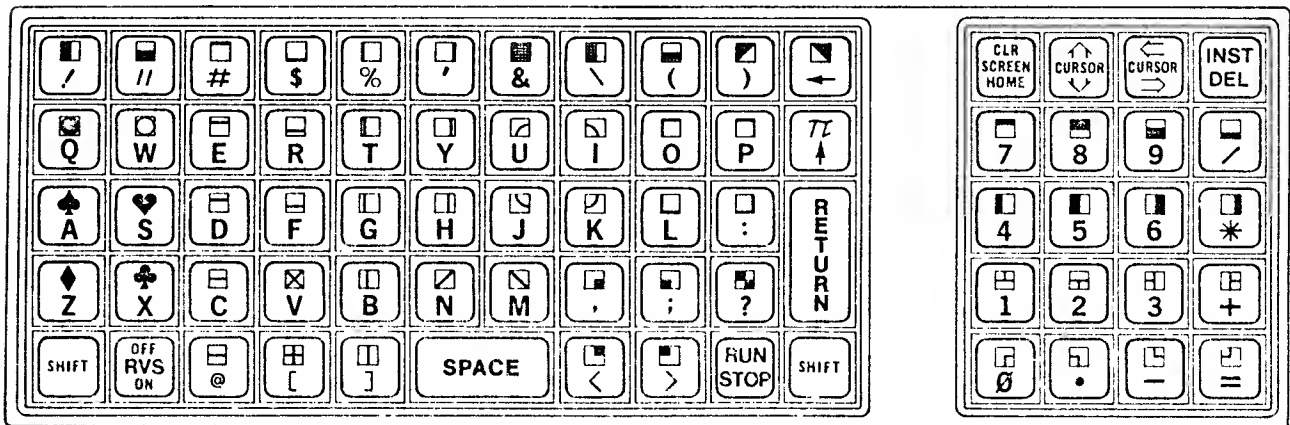


Fig. 2. The PET keyboard.



Example: Special key notation

T: 'STOP'  
R: READY.

means press the key labeled STOP.

T: STOP  
R: BREAK  
R: READY.

means press the four keys S T O P in succession.

#### F. BASIC Overview

BASIC has three classes of capabilities: commands, statements, and functions.

Commands are facilities which might be considered as part of the operating system or environment. Commands handle or manipulate global items, such as programs.

Statements are elements of the program itself. Statements are made up of keywords, variables, constants, operators, and functions.

A user defined function is a BASIC statement which returns a single value each time it is referenced. Another type of function is the library function. The PET has several library (built-in) functions, such as SIN, RND, LOG, INT, etc. These functions are referenced by their names.

1. Constants. BASIC programs are made up of statements which contain keywords, variables, operators and constants. The PET can have two kinds of constants. Numeric constants are numbers with or without a decimal point. If a number has a decimal point it is sometimes called a floating point number. A number without a decimal point is often called an integer. Floating point is used to represent a wider range of numbers than integer (see section VI on data representation for more details). Even though there are two types of numeric constants, the PET represents them the same way, i.e., as floating point. An integer constant, like 5, is represented as 5.0.

The other kind of constant that you can have in the PET is a character string. A character string constant is signaled by a quote (") mark. The symbols "AB" mean the string of two characters, A followed by B.

2. Variables. Variables are "names" that may take on different values during a problem. Variables can be named with up to 5 alphabetic characters.

## INTRODUCTION

Exercise: Determine the number of alphabetic characters permitted in a variable name.

T: ABCDE=2  
R: READY.  
T: ABCDEF=3  
R: ?SYNTAX ERROR  
R: READY.

Six or more alphabetic characters in a variable name is illegal and will give a syntax error message. If the second character in a variable name is numeric, the number of characters in the variable name is limited only by the maximum line length. X.

Exercise: Determine the number of numeric characters permitted in a variable name.

T: A123456789123456789=0  
R: READY.  
T: A1345678=5  
R: READY.

Regardless of the length of the variable, only the first two characters are used to distinguish between variables. X.

Exercise: Verify that only the first two characters are used in a variable name.

T: PRINT A1  
R: 5  
R: READY.

Notice that the variables A123456789123456789, A1345678, and A1 all refer to the same quantity. Verify that by printing out all three values.

T: PRINT A123456789123456789, A1345678, A1  
R: 5                    5                    5  
R: READY.

The first two characters are the only ones used, but additional characters can be used to describe what the variable stands for.

## GETTING STARTED WITH YOUR PET

```
T: ABCNT=33  
R: READY.  
T: PRINT AB  
R: 33
```

ABCNT could stand for the count of values A and B.

## II. PET BASIC CALCULATOR MODE

Calculator mode means that the statements entered will be executed immediately. These statements have no line numbers. The absence of a line number signals the software that this statement is to be executed immediately and not entered into the program memory. Calculator mode was used to determine the legal variable names in the preceding examples.

When typing lines, remember: each line must end with the return key pressed, T: tells you what to type, and R: tells you the response you should get from your PET.

```
T: A=5
R: READY.
T: B=10
R: READY.
T: C=A*B
R: READY.
```

These three lines made no visible effect on the PET. However, the three variables A, B, and C were set to the values specified. To check that C was calculated properly type:

```
T: PRINT C
R: 50
R: READY.
```

To check all three variables at once:

```
T: PRINT A,B,C
R: 5      10      50
R: READY.
```

Each variable in the output list is put into a field ten digits wide. There are four fields per line. The fields start in columns 2, 12, 22, and 32.

Calculator mode can be used to calculate values directly in the PRINT statement.

```
T: PRINT 5*10
R: 50
R: READY.
T: PRINT A*B+C
R: 100
R: READY.
```

## GETTING STARTED WITH YOUR PET

A shorthand for the PRINT statement in the PET BASIC is the question mark (?).

```
T: ?5*10
R: 50
R: READY.
```

### A. Using Strings in PET BASIC

```
T: A$="NAME IS"
```

Double quote is used to define the limits of a string of characters.

```
T: PRINT A$
R: NAME IS
T: B$="MY"
T: PRINT B$+A$
R: MYNAME IS
```

This concatenates the two strings A\$ and B\$ for the display only. Concatenate means to attach.

```
T: PRINT B$+"b"+A$
R: MY NAME IS
T: B$="bMYb"
T: PRINT B$+A$
R: MY NAME IS
```

To save the concatenated string.

```
T: C$=B$+A$
T: PRINT C$+"bDAVE"
R: MY NAME IS DAVE
```

### B. Numeric and Fractional Values

```
T: X=.25
T: Y=10
T: Z=X*Y
T: PRINT Z
R: 2.5
T: ? Y/X
R: 40
```

## PET BASIC CALCULATOR MODE

A ? can be substituted for the PRINT command. A / means division;  
↑ means exponentiation.

```
T: ? Y↑3
R: 1000
T: PRINT Z
R: 2.5
T: PRINT INT(Z)
R: 2
```

INT takes the integer part of a number.

### C. Conversion of Data

There are several functions available in PET BASIC to convert from one form of data representation to another. You have already seen INT which converts from a fractional number to an integer.

```
T: PRINT ASC("Z")
R: 90
```

ASC converts a character to its equivalent number; CHR\$ converts a number to its equivalent character.

```
T: L=ASC("Z")
T: PRINT L,CHR$(L)
R: 90 Z
T: L=ASC("ZXCX")
T: ?L
R: 90
```

The number 90 corresponds to the character Z. ASC only converts the first character of the string argument.

```
T: PRINT CHR$(256)
R: ? ILLEGAL QUANTITY ERROR
```

The legal characters are 0-255

```
T: PRINT CHR$(255)
R: Ɔ
T: PRINT Ɔ
R: 3.14159265
```

## GETTING STARTED WITH YOUR PET

The number 255 corresponds to the character  $\pi$  . The value of  $\pi$  is 3.14159265.

Exercise: What is the integer value of "?"? What is the character for 110?

T:

R:

T:

R:

### D. Balance Your Checkbook in Calculator Mode

Your PET can be used to help you balance your checkbook. Let the variable OB stand for the old balance.

T: OB=990.25  
T: OB=OB-12.36  
T: ?OB  
R: 977.89

This is the balance after subtracting a \$12.36 check.

T: OB=OB-10.00-20.00-22.50-11.98  
T: ?OB  
R: 913.41

This is the balance after subtracting four more checks.

T: OB=977.89  
R: READY.  
T: OB=OB-10-20-22.5-11.98  
R: READY.  
T: PRINT OB  
R: 913.41

This is the same as the previous example except that unneeded digits were not typed.

## PET BASIC CALCULATOR MODE

### E. Reserved Words

PET BASIC reserved the use of certain strings of characters to specify operations to be performed.

Exercise: Determine what the PET does with improperly used reserved words.

T: INT=3  
R: ?SYNTAX ERROR

Since INT means the integer function, it cannot be used as a variable. What would you expect from

T: ASC=1  
T: CHR\$=1  
T: PRINT=12

### F. Modes of Variables and Constants

There are three modes for variables and constants: character strings, floating point, and integer. Mixing of modes is not permissible between character strings and numeric values.

Exercise: Mixed mode (integer and floating point) arithmetic is permitted.

T: I=2.5\*10  
T: PRINT I  
R: 25



## GETTING STARTED WITH YOUR PET

Exercise: Mixing strings and numeric data is not permitted.

```
T: A=CHR$(63)
R: ?TYPE MISMATCH ERROR
```

The response is because A is a numeric variable and CHR\$ is a character variable.

Exercise: Display the numerical equivalent of the character "?".

```
T: A=ASC("?")
T: PRINT A
R: 63
```

What will you get with

```
T: D$=CHR$(63)
T: PRINT D$
R:
```

### III. INPUTTING A PROGRAM

Every program line begins with a line number. However, any statements entered without line numbers are executed immediately. Any line of text typed to BASIC that begins with a digit is processed only after a RUN command. There are four possible actions that may occur:

- X 1. A new line is added to the program. This occurs if the line number is legal (range 0-63999) for the 8K PET).
2. An existing line is modified. This occurs if the line number matches the line number of an existing line in the program. That line is modified to have the text of the newly typed line.

Exercise: Using the line editor to change a statement in your BASIC program.

T: 100 A=B+C

If you want to change "B" to "D" in line 100, one way is to

T: 100 A=D+C

3. An existing line is deleted. This happens when a number is typed that is equal to an existing line number. The line must contain only the line number, otherwise the line will be replaced and not deleted.
4. An error message (?SYNTAX ERROR) is generated. This happens if the line number is out of range (0-63999 for 8K) or the line is too long (more than 79 characters). *= twice regular*

#### A. Blanks

X Blanks preceding a line number are ignored. The first non-digit (including blanks) in a line terminates the line number. Multiple blanks are permitted anywhere in a line for readability, even in reserved words (PRINT for instance) or constants.

#### B. Multiple Program Statements

X Multiple program statements may appear on a single line if separated by a colon (:). A line number must appear only at the beginning of the first statement on the line. A memory saver is the elimination of the ending quote (") on strings since the (") requires one byte of memory. For example, PRINT "ABC works just as well as PRINT "ABC" as long as the character string is the last part of the line.

## GETTING STARTED WITH YOUR PET

However, ending quotes must be used in the following case  
100 PRINT "A=",X. To avoid ending up with programs that are hard  
to read and debug, use matching quotes even though it takes an  
additional byte. ʹ

Some examples of multiple statement lines:

```
T: NEW
T: 10 PRINT"AAA:PRINT"BBB:PRINT"CCC
T: RUN
R: AAA:PRINT 0
R: CCC
R: READY.
```

The system prints AAA:PRINT as a character string since it is contained between two quotes. It then thinks that BBB is a variable and prints the value of BBB which is 0. PRINT"CCC is the next statement and it is handled correctly. The above example shows some of the problems encountered when a quote is left out.

Exercise: The proper use of the quote in multiple statement lines.

```
T: 10 PRINT"AAA":PRINT"BBB":PRINT"CCC"
T: RUN
R: AAA
R: BBB
R: CCC
R: READY.
```

```
T: 10 A$="AB"
T: 20 PRINT A$
T: RUN
R: AB
R: READY.
```

Exercise: Multiple statements per line are also permitted in calculator mode.

```
T: FOR I=1 TO 20:PRINT I:NEXT I
```

### C. Typing Mistakes

If a typing mistake occurs during the entering of a line, two ways to correct the error are:

## INPUTTING A PROGRAM

1. Hit the 'DEL' key until the error is blanked out. Retype the rest of the line from that point.
2. Hit 'RETURN' and retype the entire line.

Other methods will be discussed in later sections.

### D. PET BASIC Commands

LIST line number - line number.

The LIST command is used to display your program. See Fig. 3. The display will be in ascending line number order regardless of the order the statements were entered. A blank is automatically inserted between the line number and the first-non-numeric character of the line even if one was not originally entered.

If a LIST command is given without a line number, all statements in memory are displayed.

If a LIST 100 is given, then line 100 only will be displayed.

If a LIST 100-150 is given, all the statements from 100 through 150 will be displayed.

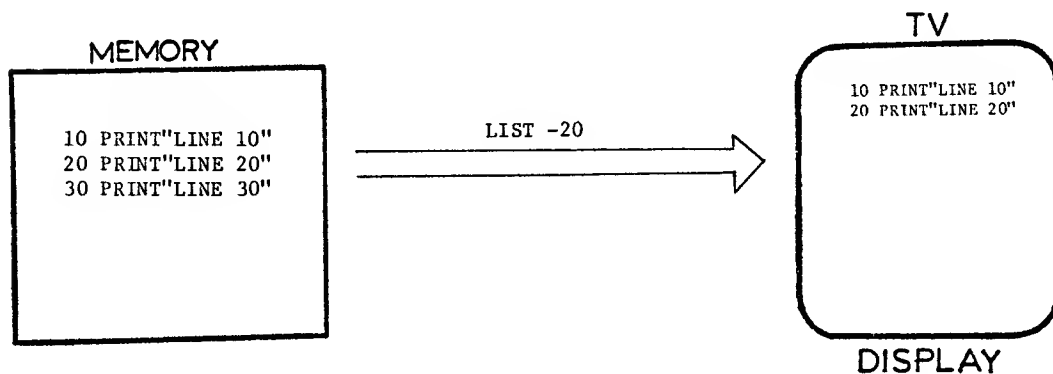


Fig. 3. LIST command action.

## GETTING STARTED WITH YOUR PET

Exercise: Use of the various options on a LIST command.

```
T: 10 PRINT"LINE 10"  
T: 30 PRINT"LINE 30"  
T: 20 PRINT"LINE 20"  
T: LIST  
R: 10 PRINT"LINE 10"  
R: 20 PRINT"LINE 20"  
R: 30 PRINT"LINE 30"  
T: LIST -20  
R: 10 PRINT"LINE 10"  
R: 20 PRINT"LINE 20"  
T: LIST 20-  
R: 20 PRINT"LINE 20"  
R: 30 PRINT"LINE 30"
```

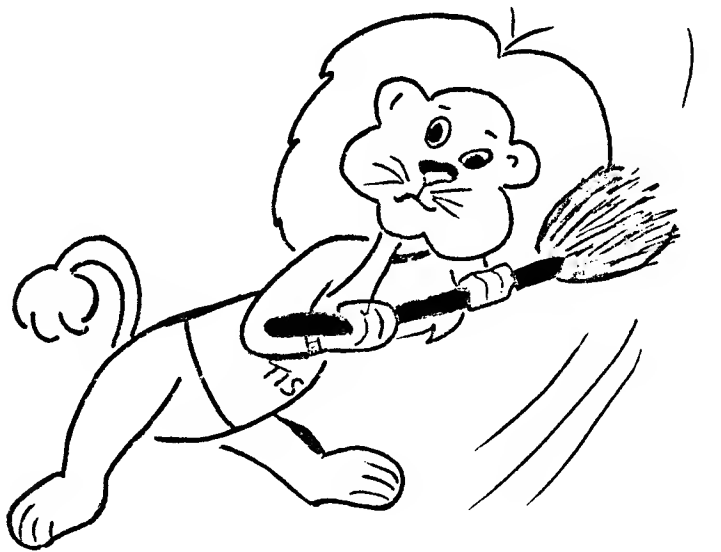


Notice that  
line 30 is  
typed out  
of order.

LIST -20 lists all statements up to and including line 20.

LIST 20- lists all statements from line 20 on.

NEW



NEW is used to reset memory before entering a new program. This is done automatically when a LOAD command is given. If a NEW command is not given and you begin to enter another program, the system cannot tell the difference between the new program and modification of the old program. If you want to keep the old program in memory, be sure to start the new program at a line number greater than the largest line number of the old program. For example:

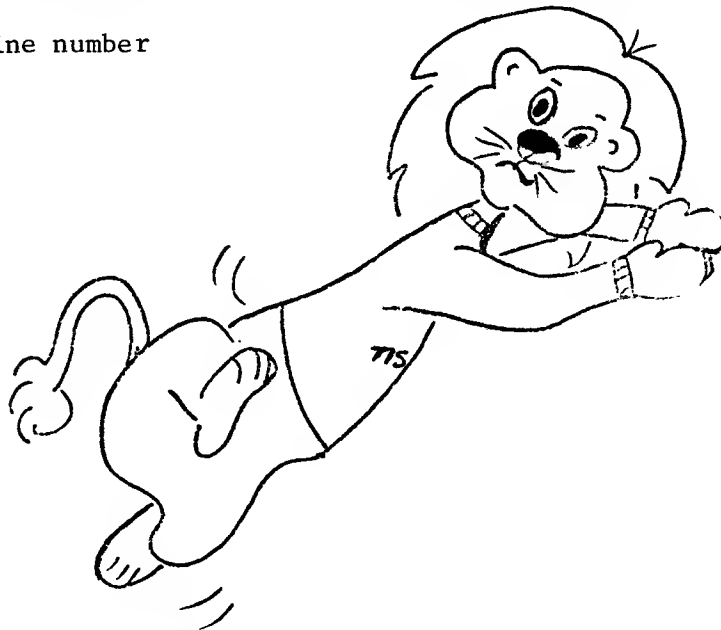
## INPUTTING A PROGRAM

Exercise: Demonstration of two programs in memory at the same time.

```
T: NEW
T: 10 R=3
T: 20 A=3.1415*R2
T: 30 PRINT A
T: 40 END
T: 100 X=π/2
T: 110 A=SIN(X)
T: 120 PRINT A
T: 130 END
```

There are now two separate programs in memory and we will soon show you how to run each one when the RUN command is discussed.

RUN line number



The line number is optional; if left out, program execution will start with the lowest line number and execute all program statements or until an END or STOP statement is encountered, whichever occurs first. If a line number is given, execution begins with that line and proceeds to execute from there.

Now let's run each of the two programs entered above.

Exercise: Selective execution of one of two programs in memory.

```
T: RUN
R: 28.2735
```

## GETTING STARTED WITH YOUR PET

This program displays the area of a circle whose radius was typed in (10).

T: RUN 100

R: 1

R: READY.

This program displays the sine of the angle (in radians),

Example: Control of execution of a program with RUN.

T: NEW  
T: 10 PRINT"LINE 10"  
T: 20 PRINT"LINE 20"  
T: 30 PRINT"LINE 30"  
T: 40 END  
T: 50 PRINT"LINE 50"  
T: 60 PRINT"LINE 60"  
T: RUN  
R: LINE 10  
R: LINE 20  
R: LINE 30  
R: READY.

RUN starts execution at the lowest line number and continues until an END or STOP is encountered or there are no more statements.

T: RUN 50  
R: LINE 50  
R: LINE 60  
R: READY.

RUN 50 begins execution with statement 50 and continues until there are no more statements.

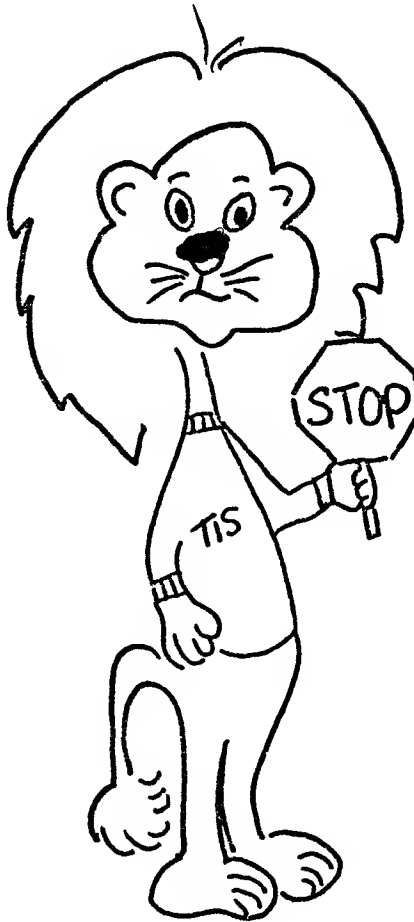
T: RUN 20  
R: LINE 20  
R: LINE 30

RUN 20 begins execution with statement 20 and stops when the END statement is encountered.

## INPUTTING A PROGRAM

### E. The 'STOP' Key and CONT

The 'STOP' key



Hitting the 'STOP' will cause the program to stop execution and will display:

BREAK IN statement number.  
READY.

The statement number will depend on when 'STOP' is typed. Typing CONT will resume execution where it left off; i.e., the next statement number.



## GETTING STARTED WITH YOUR PET

Exercise: Use of the 'STOP' key and the CONT command. Input the following program.

```
T: NEW
T: 10 I=0
T: 20 FOR A=1 TO 500
T: 30 X=A
T: 40 NEXT A
T: 44 I=I+1
T: 46 PRINT I
T: 50 GO TO 20
T: RUN
R: 1
R: 2
:
:
T: 'STOP'
R: BREAK IN 30. This line number depends on when the 'STOP' key is typed.
T: CONT
R:
```

#### IV. GETTING INFORMATION OUT OF YOUR PROGRAM

A program can calculate many different values. Unless you have some way to see the results, those calculations are useless. Results can be displayed on the TV screen in several different ways.

##### A. Output Formats - Numeric Data

Exercise: Output of uninitialized variables in field format.

```
T: NEW
T: PRINT A,B,C
R: 0      0      0
```

When commas are used to separate elements of a print list, each item is put into a separate field. There are four fields per line.

Exercise: Output of uninitialized variables in continuous format.

```
T: PRINT A;B;C
R:  0  0  0
```

When semicolons are used to separate elements of a print list, each numeric item is separated from the next by two blank spaces.

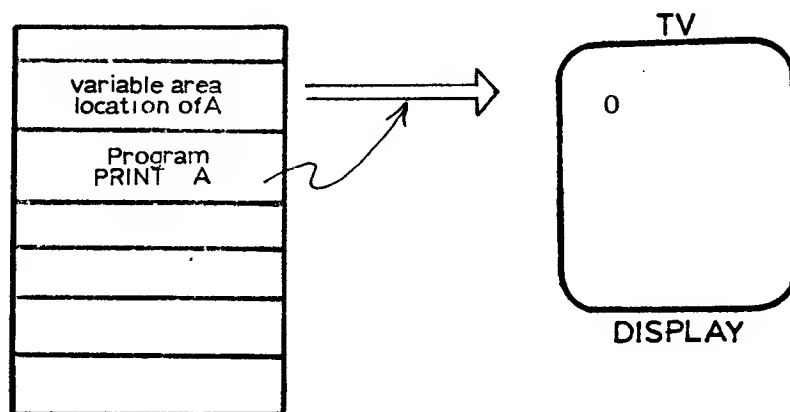


Fig. 4. PRINT action.

## GETTING STARTED WITH YOUR PET

```
T: A=123:B=456:C=789:PRINT A,B,C
R: 123      456      789
```

The colon is used to put multiple statements on one line.

```
T: A=123456789:B=1234567890:C=12345678901:
  PRINT A,B,C

R: 123456789      1.23456789E+09
   1.23456789E+10
```

Very large and very small numbers are displayed in scientific notation, and fields are skipped to make the results easier to read. In scientific notation a number is written with an exponent and a decimal point after the first digit. The exponent is the power of 10 the number is to be raised to. E is used to separate the number and the exponent.

To summarize: Display output (PRINT) statement conventions


; puts two blanks between fields and one leading blank. The 41st character in a line always appears as the first character of the next line.

, uses 10 character fields. If the number is such that the number cannot be put in the field and retain two blanks between the fields, it skips one field. Fields are not split between lines.

Numeric fields start in display columns as follows:

Field	Column
1	2
2	12
3	22
4	32 ,

Exercise: Display various values to demonstrate output conventions.



```
T: NEW
T: 100 A=1
T: 110 FOR I=1 TO 5
T: 120 A=A*10
T: 130 PRINT A,A,A
T: 140 PRINT A;A;A
T: 150 NEXT I
T: RUN
```

Notice how the numbers are displayed.

## GETTING INFORMATION OUT OF YOUR PROGRAM

```
T: 130 PRINT A,A,A,A,A
T: 140 PRINT A;A;A;A;A
```

This will display five items at one time rather than three.

```
T: RUN
```

Notice how item 5 is automatically placed on the next line.

```
T: GO TO 110
```

Note how the ; splits data between 2 lines.

### B. Output Formats - Character Strings

Exercise: String output in field and continuous formats.

```
T: NEW
T: 100 A$="A":B$="B":C$="C"
T: 110 PRINT A$,B$,C$
T: 120 PRINT A$;B$;C$
T: RUN
R: A          B          C
R: ABC
```

The comma in the print statement puts the strings in successive fields. The semicolon in the print statement puts the characters out with no spacing between them. Dies alles ist alphanumerisch gegeben

### C. Spacing

Exercise: Use of the SPC function to display blanks.

```
T: NEW
T: 200 FOR I=1 TO 40
T: 210 PRINT SPC(I);"A"
T: 220 NEXT I
T: RUN 200
```

blz 3-8  
verträglich: 205 für A = 1 to 500  
206 next A  
207 x = A

The SPC function allows you to print a variable number of spaces before displaying the data specified.

## GETTING STARTED WITH YOUR PET

Exercise: Use the TAB function to space data on the display.

```
T: 210 PRINT TAB(I);"A"  
T: RUN 200
```

The TAB function will print beginning in the column defined by the variable (I). The difference between TAB and SPC is that TAB defines the column to start printing in while SPC defines the number of spaces between fields.

Exercise: Show the difference between TAB and SPC.

```
T: NEW  
T: 10 FOR I=2 TO 10 STEP 2  
T: 20 PRINT SPC(I);"A";SPC(I);"B"  
T: 30 PRINT TAB(I);"A";TAB(I+2);"B"  
T: 40 NEXT I  
T: RUN  
R: A B  
R: A B  
R: A B  
R: A B  
.  
.  
.  
R: A B  
R: A B
```

## V. GETTING INFORMATION INTO YOUR PROGRAM

### A. Design Goals

One design goal should be to minimize the amount and type of information the person running the program must enter. The data required should be short, simple and logical. The more input required, the longer it will take and the more likely an error will occur.

In writing a program a good rule of thumb is to never use a constant where there is even the remotest possibility of a change being needed later. In those cases, use a variable instead of a constant.

On the surface, these two philosophies appear to conflict. However, BASIC provides several different methods of getting data into your program, i.e., setting variables to particular values.

The key words for getting information to your program are:

READ and DATA  
INPUT  
GET

### B. READ and DATA

READ is used to allow the programmer to easily change the value of variables (parameters) in the program. READ requires a matching DATA statement in the program.

Example: Using READ and DATA to fill a string vector. A vector is information arranged as an ordered collection of values.

```
T: NEW
T: 99 DIM MN$(12)
T: 100 FOR I=1 TO 12
T: 110 READ MN$(I)
T: 120 DATA "JAN", "FEB", "MAR", "APRIL", "MAY", "JUNE",
T: "JULY", "AUG", "SEPT", "OCT", "NOV", "DEC"
T: 130 PRINT MN$(I)
T: 140 NEXT I
T: RUN
R: JAN
R: FEB
.
.
.
R: DEC
R: READY.
```

This program sets up the array MN\$ with the names of the month and prints each one out. To verify that the fifth month is MAY try:

## GETTING STARTED WITH YOUR PET

```
T: PRINT MN$(5)
R: MAY
```

Exercise: Using READ and DATA to fill a numeric array (a collection of numbers):

```
T: 99 DIM ND(12)
T: 110 READ ND(I)
T: 120 DATA 31,28,31,30,31,30,31,31,30,31,30,31
T: 130 PRINT ND(I)
T: RUN
R: 31
R: 28
.
.
.
R: 31
```

This program puts the number of days in each month into the array ND and then prints out the number of days. To verify that the fifth month (MAY) has 31 days, try:

```
T: PRINT ND(5)
R: 31
```

### C. INPUT

INPUT allows the user of the program to provide data for the program to process. This data can be either control information to select options and specify how the processing is to be done, data to be processed, or both. A message may also be displayed at the same time input is asked for by enclosing the message in quotes following INPUT.

INPUT is used to read data from the PET keyboard. INPUT#N is for data on unit N, normally a cassette. When the INPUT statement is executed, a ? prompt is given on the TV display.

1. Floating Point Input. Floating point input allows you to enter very large or very small numbers into your PET.

Exercise: A program to explore legal floating point input data.

```
T: 4000 INPUT V:PRINT V:GO TO 4000
T: RUN 4000
R: ?
```

## GETTING INFORMATION INTO YOUR PROGRAM

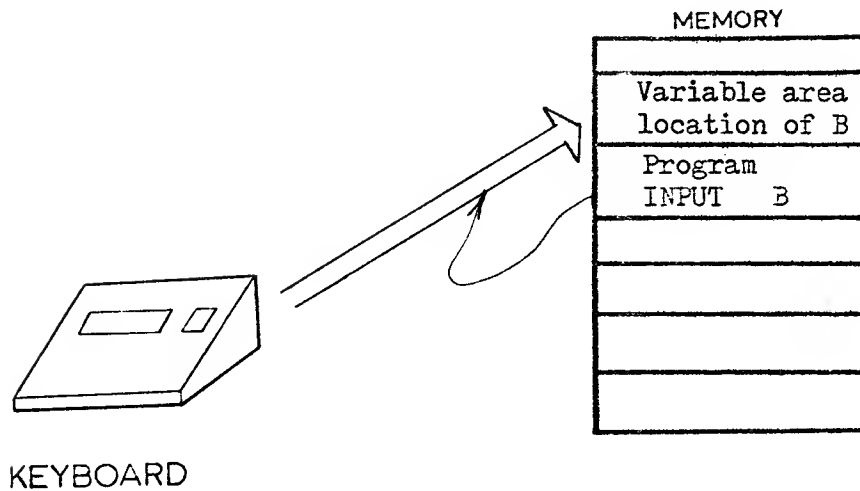


Fig. 5. INPUT action.

Since V is a floating point variable, we can use this fragment of a program to experiment with the format and size of numbers that the PET BASIC will accept. The ? implies that the program is ready for INPUT.

```
T: 1.5
R: 1.5
R: ?
```

Normal fractional numbers can be typed.

```
T: 1.5E+6
R: 1500000
R: ?
```

Scientific notation can be used for large numbers.

```
T: .005E+6
R: 5000
R: ?
T: 1235.01E+2
R: 123501
```

Scientific notation can be used to move the decimal point (scale) a number.



## GETTING STARTED WITH YOUR PET

R: ?  
T: 12.5E+40  
R: ? OVERFLOW ERROR IN 4000  
R: READY.

This message means that 12.5E+40 is too large. Earlier we saw that 1.5E+6 was not too large. What is the largest number that the PET can handle? See Section VI, Data Representation, for a small program to determine experimentally the largest and smallest numbers that the PET can represent.

The overflow error caused the program to terminate. In order to continue putting data in you must:

T: RUN 4000  
R: ?

To check the effect of leading blanks type:

T: bbb5  
R: 5  
R: ?

Try several combinations of leading, embedded, or trailing blanks to determine how they are handled by the PET.

T:    5    3    9    

R:    5,39.

R: ?

T:

R:

R: ?

T:

R:

R: ?

T: 'STOP'

R: BREAK IN 4000. This line number depends on when the 'STOP' key is hit.

## GETTING INFORMATION INTO YOUR PROGRAM

Remember that if you find a combination that generates an error message and the READY response, you must type:

T: RUN 4000

in order to continue with your input testing.

Some characters are not permitted for INPUT to a floating point variable.

T: RUN 4000

R: ?

T: 5 + 6

R: ? REDO FROM START

This error message indicates unacceptable (bad) input. It does not terminate the program. It does give you another chance. What would you expect to get if you:

T: BCD

R: ? REDO FROM START

Since the INPUT statement was expecting a floating point number, a letter (like B) would be considered unacceptable.

Try a few other symbols to determine what is legal.

T:

R:

T:

R:

T:

R:

If you didn't try something like

T: 123,456

try it now.

## GETTING STARTED WITH YOUR PET

The response:

```
R: ? EXTRA IGNORE
R: 123
R: ?
```

means that you provided more items of data than the INPUT statement requested. The comma is a data item separator.

```
T: 'STOP'
R: BREAK IN 4000. This line number depends on when the 'STOP' key is hit.
```

2. String Input. A little earlier we tried unsuccessfully to enter some non-numeric characters. The error was caused because the INPUT variable was floating point. Enter the following string INPUT test program.

Exercise: A program to explore legal string input data.

```
T: NEW
T: 2000 INPUT X$:PRINTX$:GO TO 2000
T: RUN 2000
R: ?
T: ASDE
R: ASDE
R: ?
```

Alphabetic characters are legal input for string (X\$) variables but not for floating point or integer variables.

```
T: "ASWER"
R: ASWER
R: ?
```

Note that the leading and trailing quotes are stripped off, i.e., not included in the string.

```
T: TYPE""QUOTE
R: TYPE""QUOTE
R: ?
```

All Quotes are only removed if they are at the beginning of the line.

## GETTING INFORMATION INTO YOUR PROGRAM

T: bb"AFR"  
R: AFR

Leading blanks are removed and quotes stripped off. Remember b means a blank or space.

T: bbbFGH  
R: FGH  
R: ?  
T: 'STOP'  
R: BREAK IN 2000. This line number depends on when the 'STOP' key is hit.

Leading blanks are removed. Modify your test program to see the treatment of trailing blanks.

T: 2000 INPUTX\$:Z\$=X\$+X\$:PRINTZ\$:GO TO 2000  
T: RUN 2000  
R: ?  
T: XC  
R: XCXC  
R: ?

With no blanks specified, Z\$=X\$+X\$ simply concatenates the two copies of the input string.

T: XCbb  
R: XCXC  
R: ?

Trailing blanks are removed.

T: bbbABCbbb  
R: ABCABC  
R: ?

Both leading and trailing blanks are removed.

T: bbXYZbbABbb  
R: XYZ ABXYZ AB  
R: ?

Embedded blanks are permitted.

## GETTING STARTED WITH YOUR PET

T: "bbbABCbbb"  
R:     ABC         ABC  
R:  ?

All blanks are included when enclosed in quotes.

T: "bA  
R:  A A  
R:  ?

A single set of double quotes will preserve leading blanks.

T: Ab"  
R: A "A "  
R:  ?  
T: "Abb  
R: AA  
R:  ?  
T: ABC,  
R: ?EXTRA IGNORED  
R: ABCABC  
R:  ?

A comma is treated as a data separator for string variables just as it is for numeric variables.

T: "ABC,  
R: ABC,ABC,  
R:  ?

A quote permits the comma to be part of the string and not considered as a separator.

Try several examples of using special characters in a string both with and without quotes.

## GETTING INFORMATION INTO YOUR PROGRAM

T:

R:

R: ?

T:

R:

R: ?

T:

R:

R: ?

If you haven't tried any graphics symbols, you might want to try one or two now.

*Scat good.*

T:

R:

R: ?

T:

R:

R: ?

T: 'STOP'

R: BREAK IN 2000. This line number depends on when the 'STOP' key is hit.

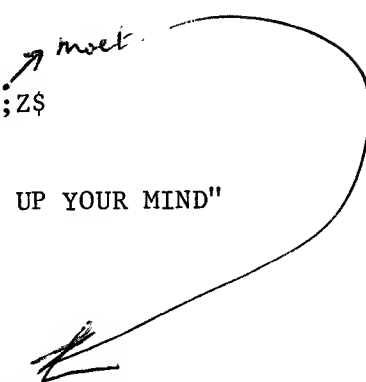
It appears that graphics symbols must be enclosed in quotes in order to be treated as string data.

Many programs use encoded responses to select options. With string handling there is no need to make encoded responses, i.e., 1 for yes, 0 for no.

## GETTING STARTED WITH YOUR PET

Exercise: Program to allow human oriented input to programs for option selection.

```
T: NEW
T: 1500 INPUT"TYPE IN YES,NO,OR MAYBE";Z$
T: 1510 IF Z$="YES" THEN PRINT "YEP"
T: 1520 IF Z$="NO" THEN PRINT "NOPE"
T: 1530 IF Z$="MAYBE" THEN PRINT "MAKE UP YOUR MIND"
T: 1540 GO TO 1500
T: RUN 1500
R: TYPE IN YES,NO,OR MAYBE?
```



Be sure to use a semicolon (;) between the last quote and the variable Z\$ in line 1500.

If you get a "syntax error" message, recheck the line number specified in the error message by:

T: LIST line number

The most likely errors are the wrong number of quotes, a misspelled keyword, an omitted \$, or a comma where a semicolon should be.

Test this program by trying:

```
T: YES
R: YEP

R: TYPE IN YES,NO,OR MAYBE?
T: NO
R: NOPE

R: TYPE IN YES,NO,OR MAYBE?
T: MAYBE
R: MAKE UP YOUR MIND

R: TYPE IN YES,NO,OR MAYBE?
T: 'STOP'
R: BREAK IN 1500
```

### 3. Integer Input

Exercise: Program to explore legal integer input data.

T: NEW  
T: 3000 INPUT I%:PRINT I%:GO TO 3000

*Neemt het gehele deel zonder af te ronden.*  
Since I% is an integer variable, we can use this fragment of a program to experiment with the format and size of numbers that the PET BASIC will accept. The ? implies that the program is ready for INPUT.

T: RUN 3000  
R: ?  
T: 1.5  
R: 1  
R: ?

Fractional parts are truncated.

T: 1.5E+6  
R: ?ILLEGAL QUANTITY ERROR IN 3000  
R: READY.

Quantity is too big.

T: RUN  
R: ?  
T: 1.5E+2  
R: 150  
R: ?  
T: bb5  
R: 5  
R: ?

Blanks are ignored.

T: 5+6  
R: ?REDO FROM START  
R: ?

This error message indicates unacceptable input. It does not terminate the program. It does give you another chance.



## GETTING STARTED WITH YOUR PET

D. GET

The GET statement differs from the READ and INPUT statements because it is conditional. That is, if data is at the keyboard (or other device specified by #), then it is put in the variable named in the GET statement. If no data is available, a special signal value (depending on the type of the variable) is put in the variable and the program is continued. GET does not wait for data.

### 1. Numeric GET.

Exercise: Experiment with numeric GET.

T: NEW  
T: 100 GET A  
T: 120 PRINT A  
T: 130 GO TO 100  
T: RUN

*verhoging for for 100 = 1 to 500  
106 106 = 1  
107 107 next i*

The screen should fill with zeros.

R: 0

*mut OFF Langzaam!*  
RVS

T: 5  
T: 5

Each time a numeric key is typed it will be displayed.

It is not necessary to type a carriage return. Watch carefully since the display will be moving rapidly. The special value returned by GET for numeric variables is the zero. Because of this, it is not possible to determine when a zero is typed.

T: 'STOP'  
R: BREAK IN 120. This line number depends on when the 'STOP' key is typed.

With the GET experiment in memory (go back and retype it if you have typed NEW or turned power off since typing it the first time).

T: 110 IF A=0 THEN GO TO 100  
T: RUN

Now nothing should happen until you type a number. Try a few:

## GETTING INFORMATION INTO YOUR PROGRAM

T:

R:

T:

R:

If you haven't tried typing zero, try it now.

T: 0

R:

What was the response? *Er gebeurt niets*

T: 'STOP'

R: BREAK IN 110. This line number depends on when the 'STOP' key is typed.

2. String GET. *gaat ook goed met meer numerieke  
sequen*  
Exercise: Determine how characters are entered via a GET  
statement.

T: NEW

T: 100 GET A\$

T: 120 PRINT A\$

T: 130 GO TO 100

T: RUN

*met off langzaam!  
rvs*

Blank lines should fill the screen. Typing any alphabetic character should echo that character as a response. Since many blank lines are being displayed, watch carefully since the character typed will quickly scroll off the top of the screen.

T: A

R: A

If you

T: 'STOP'

R: BREAK IN 120

you will interrupt the program.

## GETTING STARTED WITH YOUR PET

T: PRINT LEN(A\$)  
R: 0

When no data is provided to GET with a string variable, the length of that string is set to zero, i.e., nothing there.

T: 110 IF LEN(A\$)=0 THEN GO TO 100  
T: RUN

Now nothing should happen until you type a character. Try a few.

T:

R:

T:

R:

## VI. DATA REPRESENTATION

### A. Largest Numeric Value

1. Floating Point. To find the largest floating point number that the PET can handle, run the following program.

Exercise: Test floating point maximum.

```
T: NEW
T: 200 I=1
T: 210 I=I*2
T: 220 PRINT I
T: 230 GO TO 210
T: RUN
```

A long series of numbers starting with

```
R: 2
R: 4
R: 8
R: 16
```

will be printed out; the last number will be

```
R: 4.25352959E+37
```

followed by

```
R: ?OVERFLOW ERROR IN 210
```

This means that 8.5070 ... E+37 was too large to represent.

Exercise: Modify the program above to display large negative floating point numbers to determine the largest negative number that the PET can represent. If the sign of the result alternates between plus and minus, you should try a different modification. Your result should show that the sign is independent of the maximum size permitted.

200 wrdt i=-1.

alternend : 210 wrdt 210 i=i\*-2.

```
T: 200 i=-1.
```

```
T: run
```

```
R: ?overflow error in 210
```

# GETTING STARTED WITH YOUR PET

The largest negative number is  $-2,126,764,79 E + 37$   
 $-4,253,529,59 E + 37$

2. Integer. To find the largest integer number that the PET can handle, run the following program:

```
T: NEW
T: 300 I%=0
T: 305 I%=I%+1
T: 310 J%=2*I%-1
T: 320 PRINT J%
T: 330 GO TO 305
T: RUN
```

A series of numbers

```
R: 1
R: 3
R: 7
.
.
.
R: 32767
R: ? ILLEGAL QUANTITY ERROR IN 310
```

will be printed out. This means that 65535 is too large to represent as an integer.

```
T: I%=32768
R: ? ill. qu. Error.
```

This means that 32767 is the largest number that can be represented as an integer.

Exercise: Modify the program above to display large negative integers.

```
T: 310 J% = -2*I% + 1.
T: Run
R: -32767
R: ? ill. qu. Error in 30
T: K% = -32768
R: ? ill qu. Error
```

## DATA REPRESENTATION

The largest negative integer that can be represented by the PET is ~~-32767~~ ?

### B. Smallest Numeric Value

1. Floating Point. To find the smallest floating point number that the PET can handle, run the following program:

```
T: NEW
T: 100 I=1
T: 110 I=I/2
T: 120 PRINT I
T: 130 IF I<>0 THEN 110
T: RUN
```

A long series of numbers starting with

.5  
.25  
.125  
.  
.  
.

will be printed out the last two will be

2.93873500E-39  
0

This means that 1.469 E-39 was too small and had to be represented as a zero.

Exercise: Modify the program above to display small negative floating point numbers to determine the smallest negative number that the PET can represent.

```
T: 100 i=-1000
T: Run.
R: -2,93073500E-39
```

The smallest negative number that the PET can represent is  
----- ?

-2,93073500E-39

## GETTING STARTED WITH YOUR PET

### C. Memory Space Used

Different types of data require different amounts of memory. To determine the memory space required for each type we will use the FRE function which returns the amount of space remaining.

These are the responses for an 8K machine. If you have a 4K machine or a different version of BASIC, note the values.

Exercise: Determine the amount of space used by a single floating point value.

T: NEW  
T: DIMA(100)  
T: ? FRE(0)  
R: 6652

kies ? Fre (X) met X & IR.  
voor bytes FREE.

Determine the amount of memory space left after reserving 100 floating point locations.

T: NEW  
T: DIMA(101)  
T: ? FRE(0)  
R: 6647

Determine the amount of space left after reserving 101 floating point locations. This difference (5) is the number of bytes one floating point variable requires.

Exercise: Determine the amount of space used by a single integer value.

T: NEW  
T: DIM I%(100)  
T: ? FRE(0)  
R: 6955  
T: NEW  
T: DIM I%(101)  
T: ? FRE(0)  
R: 6953

Using the same approach you can see that for one additional integer number, two bytes are required.

## DATA REPRESENTATION

Exercise: Determine the amount of space used by a single string value.

```
T: NEW
T: DIM A$(100)
T: ? FRE(0)
R: 6854
T: NEW
T: DIM A$(101)
T: ? FRE(0)
R: 6851
```

For one additional string variable, three bytes are required. However, each string can have as many as 255 characters in it.

### D. Number of Significant Digits

*21e 612 2-3*

The PET can display nine digits in a number even though more information can be stored in memory.

Exercise: Determine the number of digits displayed for floating point numbers.

```
T: PRINT 234567890
R: 234567890
```

This indicates that nine digits can be displayed.

```
T: PRINT 1234567890
R: 1.23456789E+09
```

This shows that the tenth digit is not displayed.

Exercise: Show how many digits of information can be saved in memory.

```
T: A=1234512340:PRINT A
R: 1.23451234E+09
T: B=1234512341:PRINT B
R: 1.23451234E+09
```

Even though the numbers entered differed by 1 in the tenth digit, the display shows that they are equal. Are they really?



## GETTING STARTED WITH YOUR PET

T: IF A<>B THEN PRINT "<>"  
R: <>

The response shows that the PET can distinguish the difference of 1 in the tenth digit even if it doesn't display the difference. Can the PET distinguish a difference of 1 in the eleventh digit?

T: C=12345123400:PRINT C  
R: 1.23451234E+10  
T: D=12345123401:PRINT D  
R: 1.23451234E+10

As you expected, the displayed values appear to be equal. Are they?

T: IF C=D THEN PRINT "="  
R: =

It appears that the PET can not distinguish between two numbers that differ by one in the eleventh position.

The number of significant digits that are displayed is nine and the number saved in memory is about ten.

### E. Rounding.

Since the PET saves about ten digits in memory and only displays nine, it must "round" to display the proper value.

Exercise: Determine how the PET rounds numbers for display.

T: NEW  
T: 100 A=1234512340  
T: 110 FOR I=1 TO 15  
T: 120 B=A+I  
T: 130 PRINT A;"+";I;"=";B  
T: 140 NEXT I  
T: RUN  
R: 1.23451234E+9+1=1.23451234E+9  
:  
:  
R: 1.23451234E+9+5=1.23451235E+9  
:  
:  
R: 1.23451234E+9+15=1.23451236E+9

## DATA REPRESENTATION

✱

From this display you can see that 1234512345 was rounded to 1.23451235E+9 and 1234512355 was rounded up to 1.23451236E+9. The PET rounds the ninth digit up whenever the tenth digit is 5 (or more) and rounds down whenever the tenth digit is 4 (or less).

## VII. USING THE CASSETTE FOR PROGRAM STORAGE

### A. SAVE a Program

Since typing is so much work, you will want to save programs that you have entered. The command to do that is SAVE "LABEL".

Exercise: Write a program on the cassette.

T: SAVE  
R: PRESS PLAY & RECORD ON TAPE #1  
R: OK - This occurs after you press play and record.  
R: WRITING  
R: READY.

If you wish to give your program a label (name), type SAVE"LABEL". "LABEL" may be from 1-78 characters.

Exercise: Write a program on the tape with the label ABC.

T: SAVE"ABC"  
R: PRESS PLAY & RECORD ON TAPE #1  
R: OK - This occurs after you press play and record.  
R: WRITING ABC  
R: READY.

WARNING!WARNING! The system does not check to see which buttons are pushed on the cassette. If any button is left on, the system assumes it is the right one. Whatever button you push is assumed to be the correct one.

### B. VERIFY a Program

In order to be confident that the program you wrote on the cassette is written correctly, you may compare what is on the tape with what is in memory with the VERIFY command. First, rewind the cassette.

Exercise: Verify that the information just written is correct.

T: VERIFY  
R: PRESS PLAY ON TAPE #1  
R: OK  
R: SEARCHING  
R: FOUND  
R: VERIFYING  
R: OK  
R: READY.

## USING THE CASSETTE FOR PROGRAM STORAGE

If there is a compare error, you will get a ?VERIFY ERROR. If this happens, rewind the tape and try again. Since cassette reading and writing is not all that reliable, trying again often gets past the problem.

If a VERIFY is typed without a label, the system will try to compare the first program it finds on the cassette with what is in memory. If a VERIFY "LABEL" is typed, the system will search for the program with the given label. The name of each program encountered will be listed.

Cassettes may be removed without rewinding, then replaced, and a new program written on the cassette from that point. A 10-15 second leader is written after each SAVE command is given.

Exercise: SAVE a program and demonstrate that VERIFY catches differences. First, rewind a blank cassette.

```
T: NEW
T: 10 PRINT"TAPE TEST"
T: 20 INPUT L,W
T: 30 A=L*W
T: 40 PRINT A,L,W
T: 50 END
T: SAVE"A"

R: PRESS PLAY AND RECORD ON TAPE #1
R: OK
R: WRITING A
R: READY.
```

This puts a small program into memory and saves it on cassette with the name "A". Now rewind the cassette.

```
T: VERIFY
R: PRESS PLAY ON TAPE #1
R: OK
R: SEARCHING
R: FOUND A
R: VERIFYING
R: OK
R: READY.
```

You have succeeded in checking the information on cassette against the program in memory. Do not rewind the cassette.

## GETTING STARTED WITH YOUR PET

T: SAVE"ABC"  
R: PRESS PLAY AND RECORD ON TAPE #1  
R: OK  
R: WRITING ABC  
R: READY.

This saved a second copy of our program on the cassette. The second copy is after the first and is called "ABC".

Rewind the cassette.

To force a verify error, we will change the program in memory.

T: 10 PRINT"FORCE ERROR"  
T: VERIFY"ABC"  
R: PRESS PLAY ON TAPE #1  
R: OK  
R: SEARCHING FOR ABC  
R: FOUND A  
R: FOUND ABC  
R: VERIFYING  
R: ? VERIFY ERROR  
R: READY.

### C. LOAD a Program

To reload your program rewind the cassette and type

T: LOAD  
R: PRESS PLAY ON TAPE #1  
R: OK

The LOAD command automatically clears memory and reads into memory the first program on the cassette. The following responses are displayed after you press play on the recorder.

R: SEARCHING  
R: FOUND A  
R: LOADING  
R: READY.

If you want to load a labeled program, rewind the cassette and type LOAD"A".

## USING THE CASSETTE FOR PROGRAM STORAGE

Exercise: LOADING of programs SAVED previously.

```
T: LOAD"A"
R: PRESS PLAY ON TAPE #1
R: SEARCHING FOR A
R: FOUND A
R: LOADING A
R: READY.
```

If you have more than one program on a cassette and a LOAD is typed, the first program found is the one loaded. For example, if you have three programs on a cassette and the recorder is sitting at the end of the first program, LOAD will load the second program.

If a LOAD"ABC" is given, a search will be made for program ABC. When ABC is found it will be loaded. A list of all programs found before ABC will be displayed. For example, suppose there are four programs on a cassette, the first two are labeled 1 and \$, respectively; the third is not labeled; and the fourth is labeled PET.

If a LOAD"PET" is given, the system responds as follows:

```
T: LOAD"PET"
R: PRESS PLAY ON TAPE #1
R: OK
R: SEARCHING FOR PET
R: FOUND 1
R: FOUND $
R: FOUND
R: FOUND PET
R: LOADING
R: READY.
```

If you do not know what is on a cassette, just ask to LOAD a non-existent label and each program on the cassette will be listed as it is found.

WARNING!WARNING! Loading a program from tape is not the most reliable operation in the world. The most common error we have run into is that the system finds the label it is searching for but never successfully loads the program. The only way to correct this is to rewind the cassette and try again. Often, a second or third try will be successful.

## Notes

ASC ( " --- ")

b12 2-3

CHR\$ ( α )

Range's 3-1.

Vertraging mechanisme 3-8.

Field , ; 4-2

SPC ( α ) 4-3

TAB ( α ) 4-4.

READ / DATA 5-1.

INPUT X 5-2

INPUT X \$ 5-6

5-10

↓  
met graphics ~~getallen~~ 5-9

I <sup>02</sup> 5-11

GET A 5-12

GET A \$ 5-13

↓ met getallen.

GRENZEN 6-1 / 6-3

? FRE ( α ) 6-4.

lege opslagen display getallen 6-6

CASSETTE 7-1

# Notes



# Notes